Filed: June 30, 2003 P15915
Assignee: Intel Corporation

APPLICATION FOR UNITED STATES LETTERS PATENT

INVENTORS: Zurab KHASIDASHVILI, John MOONDANOS, Ziyad HANNA

TITLE: APPLICATION OF THE RETIMED NORMAL FORM TO THE FORMAL EQUIVALENCE VERIFICATION OF ABSTRACT RTL DESCRIPTIONS FOR PIPELINED DESIGNS

ATTORNEYS: FLESHNER & KIM, LLP
&
ADDRESS: P. O. Box 221200
Chantilly, VA 20153-1200

DOCKET NO.: Intel-022

APPLICATION OF THE RETIMED NORMAL FORM TO THE FORMAL EQUIVALENCE VERIFICATION OF ABSTRACT RTL DESCRIPTIONS FOR PIPELINED DESIGNS

BACKGROUND OF THE INVENTION

1. Field of the Invention

[1] The field of invention generally relates to formal equivalence verification tools and analyzing logic circuit models within a mathematical framework and more particularly, to an efficient method for performing sequential verification of loop-free circuits.

2. Background of the Related Art

[2] Formal Equivalence Verification (FEV) tools are used to establish the equivalence of logic circuit models by analyzing them within a mathematical framework, thereby eliminating the need to resort to computationally intensive simulation of the circuits with very large numbers of inputs.

[3] The logic design process can be viewed as a series of circuit model transformations that proceed from abstract Register Transfer Level (RTL) descriptions down to transistor level netlists. One of the factors upon which the correctness of the final, low level, implementation of the circuit depends upon, is the establishment of the equivalence of circuit models at differing levels of abstraction. Abstraction refers to the different levels of partition and hierarchy that may be used to model a circuit or a system under test. Two examples of different levels of abstraction are RTL and schematic levels.

[4] FEV tools help to establish the equivalence of logic circuit models by analyzing them within a mathematical framework, without resorting to simulating a large

number of input patterns. Frequently, the tools compare sequential logic circuit models that differ in terms of the number and placement of the state elements they contain. This is a direct consequence of the requirement to raise the level of abstraction for circuit descriptions and to improve the productivity of the front-end design activities. The performance of the classic formal equivalence verification algorithms drops significantly, as the number of state elements increases. As a result of this performance penalty, specialized algorithms have been developed that have increased performance for certain common categories of circuits. One such category of circuits includes pipelined loop-free circuits. The claimed embodiments of the present invention are concerned with the verification of pipelined loop-free circuits.

[5] The related art approaches to verification of pipelined loop-free circuits has several disadvantages. The Boolean formulas that model circuit behavior are represented using binary decision diagrams (BDDs). Therefore, these approaches suffer from the well known limitations of BDDs (e.g. memory explosion). Furthermore, related art clocking schemes are rather simplistic (see R.K Ranjan, V. Singhal, F. Somenzi, and R.K Brayton, *"Using Combinational Verification for Sequential Circuits"* in Proceedings of Design, Automation and Test in Europe Conference, 1999).

[6] Also, some related art approaches can only accommodate circuits with edge-triggered flip-flops (see G. P. Bischoff, K. S. Brace, S. Jain, and R. Razdan, *"Formal Implementation Verification of the Bus Interface Unit for the Alpha 21264 Microprocessor"* in Proceedings of IEEE International Conference on Computer Design; VLSI in Computers and Processors, 1997 and R.K Ranjan, *"Design and Implementation Verification of Finite State*

Systems", Ph.D. Thesis, Univ. of California, Berkeley, 1997). The exemplary embodiments of the present invention address many of these aforementioned problems.

[7] In the claimed embodiments of the present invention, the Boolean formulas that model circuit behavior are represented using Binary Expression Diagrams. These Boolean formulas are called Timed Binary Expression Diagrams (TBED). As a result of using this type of Boolean representation, the functionality of very complex circuits can be modeled without suffering from exponential memory requirements associated with the use of binary decision diagrams (BDDs). The comparison of formulas represented as Binary Expression Diagrams is performed with a satisfiability (SAT) solver.

[8] The combinational properties on the circuit inputs as well as on internal nodes are processed, and these properties are taken into account during the construction of TBEDs.

[9] The new algorithm for TBED computation operates in a computationally efficient manner. During a test of one exemplary embodiment of the present invention, a 120x average speedup on complex circuits with respect to older versions employing TBED construction algorithms was achieved. One reason for the increased performance is that only one traversal of the latches is needed in the verified circuits, and for each latch, two relatively simple operations (a restrict operation and a shift operation) are performed. This means that it is not necessary to compute TBEDs with non-empty event lists.

BRIEF DESCRIPTION OF THE DRAWINGS

[10] The invention will be described in detail with reference to the following drawings in which like reference numerals refer to like elements wherein:

[11] Figure 1 is a diagram of an exemplary D flip-flop with master and slave latches;

[12] Figure 2 is a block diagram of an exemplary embodiment of a method in accordance with the present invention; and

[13] Figure 3 is an exemplary system implementation of one embodiment of the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

[14] Formal equivalence verification (FEV) tools are used to establish the equivalence of logic circuit models by analyzing them within a mathematical and logical analysis framework. This avoids the technical problems, complexities and drawbacks associated with simulating the very large number of input patterns that correspond to circuits having a computationally significant level of difficulty and complexity.

[15] In many automated design verification techniques, finite state machines are used to model the behavior of systems with a finite number of states. In one exemplary scenario, the states are defined by the values of the latches in the circuit and the transitions among the various states are determined by the input and latch values. In some related art applications, successful verification occurs when the states of the circuit are successfully traversed. However, exemplary embodiments of the present invention are not based upon

state space traversal.

[16] The logic design process can be viewed as a series of circuit model transformations, as one progresses through various levels of abstraction. These levels of abstraction range from abstract resistor transistor logic (RTL) descriptions, down to transistor level netlists. This process permits a final, low level, implementation of the circuit so that equivalence of circuit models at differing levels of abstraction can be determined.

[17] The performance of the classic formal equivalence verification algorithms drops significantly as the number of state elements (e.g. latches or flip-flops) making up the circuit under test or evaluation increases. As a result, specialized algorithms have been developed that have increased performance for certain common categories of circuits. One such exemplary category of circuits includes pipelined loop-free circuits.

[18] One characteristic of a pipelined loop-free circuit is that the output of the circuit is the combinational function of the values of the inputs that flow across signal paths that are controlled by other signals.

[19] Figure 1 illustrates one exemplary embodiment of a pipelined loop-free circuit 101 that includes the master latch 102 and slave latch 104 of a D flip-flop. In Figure 1, a clock signal 106 is applied to latches 102 and 104 with an input 108 and an output 109. The following equation can be used to represent the output (Out) 109 of the circuit.

[20] Out=in[NOT Clk,Clk]

[21] This means that the output Out 109 is equal to the value of the input signal at the last time the clock signal clk was low, prior to the clock signal becoming high again. The output of a sequential circuit without feedback loops can be similarly written as:

$$\sum_i \prod_j a_{j[e_{j1},e_{j2},\ldots,e_{jk}]}^{*} \quad (1)$$

(The * sign in Expression (1) stands for a possible negation.) The terms of the form:

$$a_{j[e_{j1},e_{j2},\ldots,e_{jk}]} \quad (2)$$

[22] are known as primitive RNFs and correspond to the value of a_j at the last time signal e_{j1} became high, before signal e_{j2} became high, ..., before e_{jk} became high.

[23] The list $[e_{j1}, e_{j2}, \ldots, e_{jk}]$ is called the event list of primitive RNF (2), and a_j is called a timed Boolean variable. Expression (1) may be referred to as the Retimed Normal Form (RNF) for a sequential circuit, and circuits having equivalent RNFs may be considered as having the same functionality. In the related art, the counterparts of RNFs are called Timed (Ternary) Binary Decision Diagrams [see G. P. Bischoff, K. S. Brace, S. Jain, and R. Razdan, *"Formal Implementation Verification of the Bus Interface Unit for the Alpha 21264 Microprocessor"* in Proceedings IEEE International Conference on Computer Design, 1997] and Event Driven Boolean Functions, respectively [see R.K Ranjan, V. Singhal, F. Somenzi, and R.K Brayton, *"Using Combinational Verification for Sequential Circuits"* in Proceedings of Design, Automation and Test in Europe Conference, 1999]. However, circuits with the same functionality may possess different Retimed Normal Forms. This is possible, because the Retimed Normal Form captures the functionality of a circuit not only in terms of its

logic gates, but also of its topological structure.

[24] Figure 2 illustrates an example of a formal verification method in accordance with one exemplary embodiment of the present invention. In 200, latches are ordered in a bottom-up order. That is, state elements (e.g. latches and flip-flops) are ordered bottom-up, where the state elements that occur closest to the inputs are ordered first, and those state elements that occur closest to the output are ordered last. Therefore, in one exemplary embodiment, let $l_1, l_2, \ldots l_n$ be an ordered list of latches and assume that d_i is the data signal of l_i, and c_i is its clock signal.

[25] In 202, the following calculations are performed. For each latch l_i in the above order a TBED, tbed (l_i)is computed as follows. The TBED of its data, tbed(d_i) is calculated in accordance with line 9 of the pseudo-code algorithm shown on page 11 of the specification. The TBED of its clock tbed(c_i) is calculated in accordance with line 10 of the pseudo-code algorithm. The restriction of tbed($data_i$) over tbed(c_i) is calculated in accordance with line 10 of the pseudo-code algorithm. The TBED of latch l_i is computed by TBED(l_i) = shift $restr_i$ tbed(c_i) (line 11 of the pseudo-code algorithm).

[26] In 204, the TBED of the output is computed (see line 20 of the pseudo-code algorithm).

[27] The various circuits that will be verified are preferably represented as a Binary Expression Diagram (BED). One advantage of BEDs is that they can be compared using both BDD and SAT methods.

[28] Timed Binary Expression Diagrams (TBEDs) are RNFs where primitive RNFs are replaced by fresh different variables, called parametric variables. In transitioning

from RNFs to corresponding TBEDs, some circuit information is lost. As a result, since the claimed algorithm (and those algorithms in the related art) builds and compares TBEDs rather than RNFs, there is a possibility of false negatives occurring, because the TBEDs of equivalent circuits may not be equivalent. However, false positives should not occur, because if the TBEDs of exemplary circuits C1 and C2 are equivalent, then the circuits themselves are equivalent.

[29] These characteristics enable the construction and comparison of the TBEDs with high efficiency, and allow for a proper treatment of user given assumptions. TBEDs are computed in the format of a BED, and SAT solvers and BDDs are used to establish the equivalence of TBEDs.

[30] A BED is a representation of Boolean formulas that does not suffer from the memory explosion problem that is a characteristic of BDDs. BEDs do not have the canonicity property of BDDs. In order to check the equivalence of two BEDs, they must be translated into BDDs, or translated to another format accepted by SAT solvers.

[31] Those skilled in the art will appreciate that SAT based methods perform better than BDDs on complex Boolean formulas. A SAT engine or solver attempts to find a satisfying assignment to a Boolean formula, or concludes that such an assignment does not exist.

[32] A satisfying assignment for a formula A is an assignment of True or False to its variables, such that the value of A after the value substitution becomes True. For example, {x = True, y = False} is a satisfying assignment for formula x AND (NOT y)). SAT solvers can be used to establish the validity of a Boolean formula A, by trying to find a

satisfying assignment to the negation of A – NOT A. If and only if NOT A can be satisfied, the formula A is not valid.

[33] The TBEDs of output signals (e.g. pipelined loop-free circuits) are computed by taking into account user given properties on input as well as on internal signals. One example of the computational efficiency of one embodiment of the present invention is that TBED construction needs only one traversal over the latches. TBEDs of state elements and of the outputs are constructed in the bottom-up order, and therefore it visits each state element only once.

[34] Traditionally, the TBED of a circuit is modeled as a binary decision diagram (BDD). There are two main verification applications of BDDs: combinational verification and sequential verification. In combinational verification, the objective is to create a corresponding BDD or BDDs for the outputs of a specific design (e.g. a circuit), so that these designs can be checked for equality. In sequential verification, the objective is to represent the behavior of the finite state machine and to traverse a state-transition graph that is a representation of the state machine that the specific design is representative of.

[35] It is well known to those skilled in the art that BDDs suffer from increased memory requirements that render them inapplicable for many circuit applications (e.g. representing multiplier functionality). To overcome this limitation, TBED formulas are represented in a version of Binary Expression Diagrams (B-expressions). This is a representation that trades canonicity for smaller memory requirements. However, since this is not a canonical representation, B-expressions cannot be used to directly compare TBED formulas.

[36] Instead, the B-expressions are converted into a format accepted by SAT solvers, and SAT solvers are used to establish the equivalence of the TBED formulas. This permits the verification of designs, that would otherwise be intractable by the application of classical techniques.

[37] Another advantage of exemplary embodiments of the invention over the related art is the ability to handle combinational properties on input, as well as on internal, signals. Circuits contain properties and/or assumptions that specify the region of the Boolean space in which equivalence of a spec (specification) circuit model and an imp (implementation) circuit model must be established. Outside this region of the Boolean space, the models may differ, and this provides latitude for many optimizations during the synthesis process.

[38] TBED formulas are computed so that they reflect the logic of the assumptions made during the model development process. One exemplary way of achieving this is to use a parametric constraint solver.

[39] A parametric constraint solver receives a Boolean formula and an ordered set of variables as an input and returns an assignment to these variables that satisfies the formula. For example, given a formula ((a AND c) OR (b and d) OR (a OR c)) and a list [a,c,b,d] of variables occurring in it, a parametric solver will return the assignment [(a,a),(c,c),(b,b + !c&!a), (d,d + !c&!a)]. This assignment assigns a to a, c to c, b+!c&!a to b, and d+!c&!a to d) and this assignment satisfies the above formula. As a result, the equivalence checking is performed within the appropriate region of the Boolean space.

[40] Unlike the existing algorithms in the related art, an exemplary algorithm of the

present invention does not need to construct TBEDs with non-empty event lists, and no pending recursive calls (which consume extra memory) are needed. Two relatively simple operations (versus four or more in the existing algorithms) are needed to construct TBEDs for latches from already constructed TBEDs, and the relevant latch list is traversed only once, in a bottom-up (that is, from inputs toward the outputs) order.

[41] One example of pseudo-code for implementing the TBED construction algorithm appears below.

```
1. Compute tbed(n) {
2.     / / This part computes TBEDs of all relevant latches
3.  ordered_latches : = order_bottom_-up (cone_latches)
4.  remaining_-latches := ordered_latches
5.  WHILE {remaining_latches is not empty}
6.    latch := head remaining_latches
7.    remaining_-latches := tail remaining_-latches
8.    IF{latch is latching data =G(l-1 ,...,l-m) on active control ctrl = H(l'-1 .. .,l'-k)}
9.      tbed(data) := G(tbed(l1),..., tbed(lm))
10.     tbed(ctrl) := H(tbed(l' -1),..., tbed(l'k))
11.     restr := restrict tbed(data) tbed(ctrl)
12.     tbed(latch) : =shift restr tbed(ctrl)
13.   ENDIF
14. ENDWHILE
15.   // Now it remains to compute the TBED of the output n
16. IF {n=O(l*-1, ..., l*-r)}
17.   IF {n is input}
18.   return P(n[]);
19.   ELSE
20.   return O(tbed(l*1 ) ... tbed(l*r))
21.   ENDIF
22. ENDIF
23. }
```

[42] First of all, the exemplary pseudo-code orders the latches in a bottom up order which moves from the inputs to the outputs. The latches are ordered in a bottom up

manner to compute TBED of a latch or a logic gate, TBEDs of the latch pins (e.g a data pin) or TBEDs of the inputs of the logic gate respectively. Computing the TBEDs in a bottom-up order helps to ensure that when a TBED of a circuit node is computed, all the needed TBEDs are already computed.

[43] The exemplary algorithm traverses the ordered list of latches by always examining its leftmost element, which is also called the head of the list. The algorithm builds the TBED of that latch, removes it from the list, and moves on to the next element, which is the head of the remaining list.

[44] In lines 9 and 10 of the pseudo-code, TBEDs of the pins of the currently examined latch is computed. In lines 11 and 12, the restriction and shift operation is applied to compute the TBED of the currently examined latch.

[45] In line 20, once the TBEDs of all latches are computed after traversing the ordered latch list, the TBED of the output is computed by applying Boolean operations to the TBEDs of some of the computed latches. Line 18 refers to a degenerate case of a circuit which consists of an input only and does not contain any logic gates or latches. The restrict is an adaptation of a standard restrict operation that is applied on BDDs

[46] P( ) is a one to one mapping between primitive RNFs and a special set of parametric variables. Parametric variables are ordinary variables that are only used for the purpose of denoting primitive RNFs. The names of the parametric variables are chosen so that a node with the same name does not occur in the circuits that are to be verified. This is done so that the probability of false negatives is minimized, and the same parametric variable denotes the same or equivalent primitive RNFs. The shift operation updates

parametric variables by inserting new events into the event lists of the corresponding primitive RNFs.

[47] One application of the exemplary embodiments of the present invention is the verification of microprocessors and the pipelined circuits that make up these microprocessors. The increased performance and capacity of the disclosed algorithm enables the verification of complex pipelined circuits within formal mathematical frameworks. This helps to avoid or reduce the time-consuming simulation of input patterns on both specification and implementation models. The various embodiments of the present invention permit the ability to more efficiently establish that the logic behavior of larger and more complex pipelines actually complies with their corresponding specifications. This helps avoid the costly design errors and testing problems that delay the production of healthy, well-functioning processors.

[48] At various stages of microprocessor design, the microprocessor is modeled at different levels of abstraction. To verify the equivalence of all or parts of two microprocessor designs using the claimed embodiments of the present invention, the designs are represented at the gate level. For example, gate level netlists can be produced from RTL or schematic representations of the design. Two gate level representations are compared by a tool implementing the claimed verification method and algorithm.

[49] Figure 3 illustrates an exemplary global illustration of a computer incorporating a system for performing FEV of pipelined designs. The computer system may include a microprocessor 300, which includes many sub-blocks, such as an arithmetic logic unit (ALU) 302 and an on-die cache 304. Microprocessor 300 may also communicate to

other levels of cache, such as an off-die cache 306. Higher memory hierarchy levels such as system memory 308 (e.g. RAM), are accessed via a host bus 310 and chipset 312. In addition, other off-die functional units, such as a graphics accelerator 314, a network interface 315 and a modem 318, to name just a few, may communicate with the microprocessor 300 via appropriate busses, ports or other communication devices.

[50] In Figure 3, a formal equivalence verification system 320 is connected to the microprocessor or may be connected to gate level netlists that represent the design. Note that the formal equivalence verification system 320 and its interface to the item under test is exemplary in nature and may be interfaced or coupled with the computer system or item under test in various different configurations and locations without departing from the spirit and scope of the embodiments of the invention.

[51] In another embodiment of the present invention, a computer readable media that implements an exemplary method of the present invention is disclosed. There is code that lists the latches in a predetermined order, code that represents the latch functions in a binary format and code that computes a Timed Binary Expresssion Diagram (TBED) using a binary format. Those skilled in the art will appreciate that this computer readable media may be implemented on any form of media (or combination thereof), such as a hard drive, CD, floppy disk, magnetic memory etc.

[52] The foregoing embodiments and advantages are merely exemplary and are not to be construed as limiting the present invention. The present teaching can be readily applied to other types of apparatuses. The description of the present invention is intended to be illustrative, and not to limit the scope of the claims. Many alternatives, modifications,

and variations will be apparent to those skilled in the art. In the claims, means-plus-function clauses are intended to cover the structures described herein as performing the recited function and not only structural equivalents but also equivalent structures.